OmpSs and the Nanos++ runtime

E. Ayguade (eduard.ayguade@bsc.es), R. Badía, J. Labarta
Barcelona Supercomputing Center(BSC)/Technical University of Catalonia (UPC)

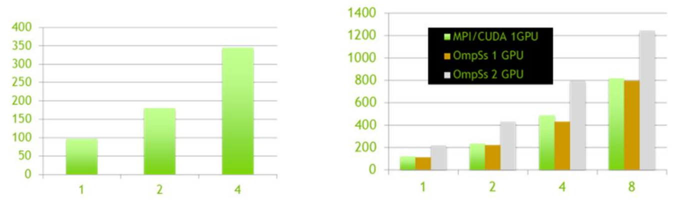
This position paper proposes the OmpSs programming model and the Nanos++ run time as potential components for the exascale software stack.

The OmpSs [5] programming model extends the OpenMP task directives with a unified mechanism to enable the exploitation of intertask dependencies and locality aware scheduling policies by the runtime. Data access directionality clauses (in, out, inout) for tasks provide the information required by the system to support dependencies and locality optimizations. The main logic of the program that orchestrates the chaining of task is in itself a valid sequential program form which the system extracts the potential concurrency and data access or movements required. The programming model provides the right decoupling between programs and systems, letting developers focus on algorithms and science and the runtime on optimizing the usage of the resources and how the programs are mapped to them.

OmpSs aims at an elegant single source solution for programmers to achieve very good performance of a wide variety of system architectures. When targeting heterogeneous systems, several implementations for a given task can be provided by the programmer or used from libraries. The system can dynamically choose the appropriate task implementations and schedules, adapting to the changes in application characteristics and resources availability and performance.

OmpSs is a “node” level programming model that nicely integrates into MPI in a hybrid programming approach [9]. MPI+OmpSs has the potential to extend to the global extreme scale execution the asynchronous model that it implements within a node, This naturally provides a huge potential for overlap between communication and computation and to highly relax the often too synchronous structure of MPI applications. Besides enabling higher overall efficiency (tolerance to latency and bandwidth limitation) it is also a very useful mechanism to fight OS and network noise. Further integration with other cluster level models such as other PGAS models should also be considered. Many of the same issues would also appear, but using some of the additional flexibility they provide in some of the solutions would be beneficial.

Different implementations of the Nanos++ runtime have been developed targeting different platforms. They allow the execution of the single address space user level model on shared memory nodes (SMP/NUMA) on nodes with one or multiple accelerators (GPUs[1], work ongoing on FPGA) and even small clusters each of them with multicore and accelerator based nodes [8].



(a) (b)

Figure 1: Scalability (Kparts/s) of the same OmpSs nbody code run on (a) a node with one to four GPUs (TESLA 2050) and (b) a cluster with upt to 8 nodes each of them equiped with two GPUs (TESLA 2090).

Figure 1 shows how a single source code of an nbody simulation scales on different target configurations. The source code has been derived from the nbody example un the NVIDIA distribution by eliminating all CUDA memory allocation and data transfer, leveraging the CUDA kernel and annotating with OmpSs pragmas a loop that applies the kernel to blocks of particles. The resulting OmpSs code has 11 lines (not counting the kernel) while the corresponding part in the original CUDA code has 14 lines. The resulting OmpSs code can run and scales on multiple GPUs within a single node or a cluster with GPUs on each node while the original CUDA code can only run in one GPU

Extensive ongoing research is ongoing on the model itself (pragmas) and how it fits real large scale applications as well as on the runtime. In particular looking at implementations targeting different platforms (from general purpose muticores to different accelerators and FPGAs, integration of CUDA, OpenCL and OpenACC), different scheduling plugins in particular looking at locality optimization and dynamic auto tuning selecting the most appropriate among multiple implementations of a task. Work is also done on fine grain load balance of MPI+OmpSs applications[10], where the runtimes of different processes in the same node coordinate and interact with the underlying OS kernel to dynamically migrate cores between processes when synchronizing MPI calls are reached.

Challenges addressed: The OmpSs jointly addresses programmability/portability and performance at very large scale by enabling the efficient exploitation of multicore nodes (whose growth is a main contributor to the global growth in scale required to address the exascale).

Maturity: The OmpSs model is being developed at the Barcelona Supercomputing Center and supersedes all previous developments under the generic name of StarSs (CellSs[1], SMPSs[2], GPUSs[4]) integrating in a single infrastructure the techniques for which evidence on their usefulness has been gained over the past 6 years. The references section enumerates some of the publications during this period. A fairly stable structure is available, being used for large scale scientific applications within the Text, Montblanc and DEEP projects of the EU exascale call. The infrastructure is open source and accessible from http://pm.bsc.es.

Uniqueness: OmpSs originates from the vision that the extreme needs of very large scale systems pose. Such needs drive the deep focus of OmpSs on the fundamental issues such as how to convey to the runtime abstract information that describes the structure, characteristics and real needs of the algorithms/programs without being target machine specific.

Novelty: Research work leading to the OmpSs proposal and infrastructure started at BSC around 2004 and still there is a whole lot of potential to explore both in terms of definition of the programming model interface and in the development of intelligence in the runtime. The ideas being used draw form the superscalar processor design and data flow research. The big novelty of the model is to make those functionalities available for programmers in an environment very similar to what they are used to (a sequential imperative language plus directives) and to enable the application of techniques developed in the past under totally new scales and constraints. For example, instead of being limited to 200 instructions in flight or renaming a few tenths of registers as superscalar processors do, the OmpSs runtime can handle 20000 or more tasks in flight or rename megabytes of data. Several orders of magnitude increase in the scale of what can be done certainly motivates new solutions. Revolution can be as simple as reconsidering old ideas under totally new eyes and scales.

Applicability: Even if originating from HPC and exascale needs, OmpSs aims at providing general purpose solutions for multicore programming. Thus OmpSs will be useful at other scales (petascale departmental, personal laptop or even mobile devices as multicores are now pervasive) and application areas (data analytics, non numeric, gaming, …).

Effort: Developing a programming model and runtime is a continuous and daunting effort. Different teams can contribute both on the compiler and runtime side. Basic development and maintenance teams might require around 5 engineers for compiler and 8 engineers for the different runtime topics and targets.

References

1. P. Bellens, J. M. Perez, R. M. Badía, J. Labarta “CellSs: A programming Model for the Cell BE Architecture” SC 2006 (Tampa).
2. J. M. Perez, R. M. Badia, and J. Labarta. “A dependency-aware task-based programming environment for multi-core architectures”. Cluster 2008, September 2008.
3. P. Bellens, J. M. Perez, F. Cabarcas, A. Ramirez, R. M. Badia and J. Labarta. “CellSs: Scheduling Techniques to Better Exploit Memory Hierarchy”. Scientific Programming, vol 17, pp. 77-95, January 2009.
4. E. Ayguade, R. M. Badia, F. D. Igual, J. Labarta, R. Mayo and E. S. Quintana-Ortí. “An Extension of the StarSs Programming Model for Platforms with Multiple GPUs”. EuroPar 2009, August 2009
5. E. Ayguade, R. M. Badia, P. Bellens, D. Cabrera, A. Duran, M. Gonzalez, F. Igual, D. Jimenez-Gonzalez, J. Labarta, L. Martinell, X. Martorell, R. Mayo, J. M. Perez, J. Planas and E. S. Quintana-Ortí. “Extending OpenMP to Survive the Heterogeneous Multi-core Era”. Intnl. Journal of Parallel Programming, vol. 38, no: 5-6, pages: 440-459, June 2010.
6. R. Ferrer, J. Planas, P. Bellens, A. Duran, M. Gonzalez, X. Martorell, R. Badia. E. Ayguade, J. Labarta. “Optimizing the Exploitation of Multicore Processors and GPUs with OpenMP and OpenCL”. LCPC2010, October 2010.
7. J. M Perez, R. M. Badia and J. Labarta. “Handling task dependencies under strided and aliased references”. 24th ACM International Conference on Supercomputing, J Tsukuba, Japan, June 2010.
8. J. Bueno-Hedo, J. Planas, A. Duran, R.M. Badia, X. Martorell, E. Ayguadé and J. Labarta. “Productive Programming of GPU Clusters with OmpSs”. IPDPS 2012, Shanghai, May 2012.
9. V. Marjanovic, J. Labarta, E. Ayguade and M. Valero. ”Overlapping Communication and Computation by Using a Hybrid MPI/SMPSs Approach”. 2ICS 2010. Tsukuba, June 2010.
10. M. Garcia et al. “LeWI: A Runtime Balancing Algorithm for Nested Parallelism”. ICPP09